

Software Testing - Summer 2024

Software Testing Process

Steps

- 1. **Analysis:** Analyze specification and software implementation
- 2. **Identify Test Coverage Items:** Criteria the tests address
- 3. **Identify Test Cases:** Specify data required to address test coverage items (TCI)
- 4. **Test Design Verification:** Review test cases to ensure every TCI has been covered
- 5. **Test Implementation:** Implement using test tool library
- 6. **Test Execution:** Execute tests using selected input data values
- 7. **Review Test Results:** Examine test results for failures

Test Cases Template

ID	TCI	Inputs (multiple columns)	Expected Results (return value)
ID	List	Values	Values

For error values the ID and the error value get marked with a *

Black Box Testing

Testing for specification: does the tested code generate correct results?

Equivalence Partitions

- 1. **Analysis:** Identify natural ranges & specification-based ranges (Specification-based ranges declare which range changes the result)
- 2. **Identify Test Coverage Items:** TCI for each partition of each input / output parameter
- 3. **Equivalence Value for each TCI:** Identify middle value of short ranges or convenient value for long ranges
- 4. **Identify Test Cases:** Select first uncovered TCI, indicate [duplicates] when all TCIs of input are covered or result is already covered, last Test Cases cover error partitions, indicated with *
- 5. **Test Design Verification:** Check if all TCIs are covered

Examples

TCI	Parameter	Equivalence Partition	Test Case
EP1*	bonusPoints	Long.MIN_VALUE..0	T1.4
EP2		1..80	T1.1
EP3		81..120	T1.2
EP4		121..Long.MAX_VALUE	T1.3
EP5	goldCustomer	true	T1.1
EP6		false	T1.2
EP7	Return Value	FULLPRICE	T1.1
EP8		DISCOUNT	T1.3
EP9		ERROR	T1.4

ID	TCI Covered	Inputs		Exp. Results
		bonusPoints	goldCustomer	return value
T1.1	EP2,5,7	40	true	FULLPRICE
T1.2	EP3,6,[7]	100	false	FULLPRICE
T1.3	EP4,[6],8	200	false	DISCOUNT
T1.4*	EP1*,9	-100	false	ERROR

Boundary Value Analysis

1. **Analysis:** Same as with EPs
2. **Identify Test Coverage Items:** Each TCI covers one boundary value (min or max)
3. **Identify Test Cases:** Same as with EPs

Examples

TCI	Parameter	Boundary Value	Test Case
BV1*	bonusPoints	Long.MIN_VALUE	T2.7
BV2*		0	T2.8
BV3		1	T2.1
BV4		80	T2.2
BV5		81	T2.3
BV6		120	T2.4
BV7		121	T2.5
BV8		Long.MAX_VALUE	T2.6
BV9	goldCustomer	true	T2.1
BV10		false	T2.2
BV11	Return Value	FULLPRICE	T2.1
BV12		DISCOUNT	T2.5
BV13		ERROR	T2.7

ID	TCI Covered	Inputs		Exp. Results
		bonusPoints	goldCustomer	return value
T2.1	BV3,9,11	1	true	FULLPRICE
T2.2	BV4,10,[11]	80	false	FULLPRICE
T2.3	BV5,[10,11]	81	false	FULLPRICE
T2.4	BV6,[10,11]	120	false	FULLPRICE
T2.5	BV7,[10],12	121	false	DISCOUNT
T2.6	BV8,[10,12]	Long.MAX_VALUE	false	DISCOUNT
T2.7	BV1*,13	Long.MIN_VALUE	false	ERROR
T2.8	BV2*,[13]	0	false	ERROR

Decision Table

1. **Analysis:** Create boolean expressions for each non-error partition
1. Identify the boolean expressions

2. Fill decision table

3. Remove impossible columns

4. Fill effect / return value

5. Verify each rule
2. **Identify Test Coverage Items:** Each rule is a TCI
3. **Identify Test Cases:** Each rule has a candidate test case, duplicates get removed
4. **Test Design Verification:** Fill TCI table, check for duplicates, every TC covers at least 1 TCI

Examples

		Rules					
		1	2	3	4	5	6
Causes	bonusPoints ≤ 80	T	T	F	F	F	F
	bonusPoints ≤ 120	T	T	T	T	F	F
	goldCustomer	T	F	T	F	T	F
Effects	return value == FULLPRICE	T	T	F	T	F	F
	return value == DISCOUNT	F	F	T	F	T	T

ID	TCI Covered	Inputs		Exp. Results
		bonusPoints	goldCustomer	return value
T3.1	DT3	100	true	DISCOUNT
T3.2	DT5	200	true	DISCOUNT

TCI	Rule	Test Case
DT1	1	T1.1
DT2	2	T2.2
DT3	3	T3.1
DT4	4	T1.2
DT5	5	T3.2
DT6	6	T1.3






White Box Testing

Testing the implementation: do valid results get generated when executed?

Statement Coverage

1. **Analysis:** Run existing tests, check JaCoCo coverage report for missed statements
2. **Identify Test Coverage Items:** Create TCI to reach uncovered statement
3. **Identify Test Cases:** Create TC for every TCI
4. **Test Design Verification:** Complete TCI table, remove duplicates, every TCI must cover at least 1 TC

Examples

```
21.      
22.      public static Status giveDiscount(long bonusPoints, boolean goldCustomer)
23.      {
24.          Status rv = FULLPRICE;
25.          long threshold=120;
26.
27.           if (bonusPoints<=0)
28.              rv = ERROR;
29.
30.          else {
31.               if (goldCustomer)
32.                  threshold = 80;
33.               if (bonusPoints>threshold)
34.                  rv=DISCOUNT;
35.          }
36.
37.           if (bonusPoints==43) // fault 4
38.              rv = DISCOUNT;
39.
40.          return rv;
41.      }
```

ID	Line Number	Condition
1	38	bonusPoints==43

TCI	Line	Test Case
SC1	38	T4.1

ID	TCI	Inputs		Exp. Results
		bonusPoints	goldCustomer	return value
T4.1	SC1	43	false	FULLPRICE

Branch Coverage

Examples

```
22. public static Status giveDiscount(long bonusPoints, boolean goldCustomer)
23. {
24.     Status rv = FULLPRICE;
25.     long threshold=120;
26.
27.     if (bonusPoints<=0)
28.         rv = ERROR;
29.
30.     else {
31.         if (goldCustomer && bonusPoints!=93) // fault5
32.             = 80;
33.         if (bonusPoints>threshold)
34.             rv=DISCOUNT;
35.     }
36.
37.     return rv;
38. }
```

Branch	Start Line	End Line	Condition
B1	31 (branch 3)	33	goldCustomer && bonusPoints==93

After implementing Statement an Branch Coverage Tests check JaCoCo report!

Object Oriented Testing

Refers to testing methods in the context of their class, as methods interact with each other.

1. **Analysis:** Categorize class methods, select testing technique
 - 1. *Static methods:* usually no testing (?)
 - 2. *Constructors:* Test attribute initialization
 - 3. *Accessor methods:* Only test if they have more than a single assignment / return statement
 - 4. *Methods with no class interaction:* Test using non-OO techniques
 - 5. *Methods with class interaction:* Test all of these
2. **Identify Test Coverage Items:** All methods to test, including in- and outputs and euqivalence partitions
3. **Identify Test Cases:** TC for each TCI, consists of multiple method calls
4. **Test Design Verification:** All TCIs covered, no duplicates

Examples

Analysis

Constructor	Attribute	Value
SpaceOrder(x:bool)	special accept	true or false (as passed to the constructor) false (specified in the class diagram)

Attribute	Getters	Setters
special	getSpecial()	–
accept	getAccept()	–

Method	Read	Written
acceptOrder(int)	special	accept

TCs / TCIs

ID	TCI Covered	Inputs	Expected Results Return Value
T1	EP1 EP3 EP5	new SpaceOrder(true) getSpecial() getAccept()	true false
T2	EP2 EP4	new SpaceOrder(false) getSpecial()	false
T3	[EP1] EP7,10,14 EP12	new SpaceOrder(true) acceptOrder(7) getAccept()	true true
T4	[EP2] EP8,11,[14] [EP12]	new SpaceOrder(false) acceptOrder(504) getAccept()	true true
T5	[EP2] EP9,[10,14] EP13	new SpaceOrder(false) acceptOrder(5000) getAccept()	true false
T6*	[EP2] EP6,15 [EP13]	new SpaceOrder(false) acceptOrder(-5000) getAccept()	false false

TCI	Method	Name	Equivalence Partition	Test Case
EP1	SpaceOrder()	isSpecial	true	T1
EP2			false	T2
EP3		special	true	T1
EP4			false	T2
EP5		accept	false	T1
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0	T6
EP7			1..15	T3
EP8			16..1024	T4
EP9			1025..Integer.MAX_VALUE	T5
EP10		special	true	T3
EP11			false	T4
EP12		accept	true	T3
EP13			false	T5
EP14		return value	true	T3
EP15			false	T6

Application Testing

- Analysis:** Identification of site structure
 - Analyze different screens
 - Analyze user interface elements on each screen
 - Analyze how input and output data is represented
- Identify Test Coverage Items:** Each user story with each acceptance criterion, select suitable values
- Identify Test Cases:** each TCI has one TC that checks the output
- Test Design Verification:** Review each test case to its user story / acceptance criterion
- Test Implementation:** Something like Selenium can be used for web app interaction

Analysis

Page Title	HTML Element/Type	id
Fuel Checker	<input type="text"/> <input type="checkbox"/> <input type="button"/> <input type="button"/> <a> 	litres highsafety Enter Info exitlink
Fuel Check Information	<input type="button"/> <div> </div>	goback body
Results	<input disabled="" type="text"/> <input disabled="" type="checkbox"/> <input type="button"/> <input type="text"/> <a> 	litres highsafety Continue result exitlink
Thank you	<div> </div>	body

TCI	Output	Value
US1	result	"Fuel fits in tank."
US2	result	"Fuel fits in tank."
US3	result	"Fuel does not fit in tank."
US4	result	"Fuel does not fit in tank."
US5	body	contains "Standard tank capacity: 1200 litres" and "High safety tank capacity: 800 litres"
US6	body	"Thank you for using fuelchecker"
US7	result	"Invalid data values."

ID	TCI Covered	Inputs	Expected Results
T1	US1	Enter "1000" into litres Deselect highsafety Click on Enter	Moved to Results screen Result is "Fuel fits in tank."
T2	US2	Enter "400" into litres Select highsafety Click on Enter	Moved to Results screen Result is "Fuel fits in tank."
T3	US3	Enter "2000" into litres Deselect highsafety Click on Enter	Moved to Results screen Result is "Fuel does not fit in tank."
T4	US4	Enter "1000" into litres Select highsafety Click on Enter	Moved to Results screen Result is "Fuel does not fit in tank."
T5	US5	Click on Info	Moved to Information screen body contains "Standard tank capacity: 1200 litres" body contains "High safety tank capacity: 800 litres"
T6	US6	Click on exitlink	Moved to Thank you screen body contains "Thank you for using fuelchecker."
T7	US7	Enter "xxx" into litres Select highsafety Click on Enter	Moved to Results screen Result is "Invalid data values."

TCI	Acceptance Criteria	Test Case
US1	S1A1	T1
US2	S1A2	T2
US3	S1A3	T3
US4	S1A4	T4
US5	S1A5	T5
US6	S1A6	T6
US7	S1A7	T7

TCI	Input	Value
US1	litres	"1000"
US2	litres	"400"
US3	litres	"2000"
US4	litres	"1000"
US7	litres	"xxx"

Test Automation

Can be achieved through different frameworks. Usually tests get run on every pull / merge request through the use of a CI pipeline.

Path Testing

1. **Analysis:** Identify all possible paths through the application, remove logically impossible paths
2. **Test Cases and Coverage Items as in other black- and whitebox testing techniques**

Random testing


A large number of inputs gets selected randomly, approximates to exhaustive testing.
Exercises program, demonstrates that it does not crash.

3 Problems:

- **Test Completion Problem:** No information on how many tests are needed
- **Test Data Problem:** Poor distribution of inputs, as it doesn't respect partitions
- **Test Oracle Problem:** Can't tell if results are correct -> too many test

Proceed as with Equivalence Partitions, but take random values for partitions:

1. **Identify Test Cases:** Random input between boundaries of value

ID	TCI Covered	Inputs 		Exp. Results
		bonusPoints	goldCustomer	return value
T12.1	EP2,5,7	rand(1,80)	true	FULLPRICE
T12.2	EP3,6,[7]	rand(81,120)	false	FULLPRICE
T12.3	EP4,[6],8	rand(121,Long.MAX_VALUE)	false	DISCOUNT
T12.4*	EP1*,9	rand(Long.MIN_VALUE,0)	false	ERROR

1. **Test Design Verification:** If test fails, break and report error